Schema Evolution in Federated Information Systems

Susanne Busse

TU Berlin, FB Informatik
Einsteinufer 17
D-10587 Berlin, Germany
sbusse@cs.tu-berlin.de

Claudia Pons

Lifia-UNLP
Calle 50 esq.115 1er.Piso
(1900) La Plata, Argentinia
cpons@info.unlp.edu.ar

Abstract

Mediators – used in federated information systems – provide a homogeneous read-only access to a set of autonomous information sources. To achieve semantic integration of the heterogeneous data, correspondences between the autonomous schemas are specified manually by experts. Considering the continuous evolution of those typically long-living information systems, schema evolution is an important aspect. New concepts are necessary to adapt correspondences consistently to evolving schemas.
In this paper we propose a formalized schema evolution mechanism for federated information systems based on metamodeling and dynamic logic. We define useful transactions for schema evolution based on three elements: (1) a metamodel for ODMG schemas, (2) a metamodel for model correspondence assertions (MoCAs), and (3) a classification of evolution actions and their specification. For each evolution transaction, semantic preconditions are specified and the impact on the correspondence assertions are described. In this way, we contribute an important step towards a consistent evolution of federated information systems.

1 Introduction

1.1 Federated Information Systems

The integration of autonomous information systems into one information infrastructure is a well known problem in modern information systems engineering ([SL 90], [Wie 92]). We call those infrastructures federated information systems (FIS, [BKLW 99]).

We address tightly coupled FIS, where a mediator provides a homogeneous read-only access point to a dynamically changing information base of autonomous, heterogeneous, and distributed legacy information systems. The mediator uses a virtual global schema for semantic integration of the data sources. Wrappers encapsulate data sources resolving technical differences and heterogeneous data models.

To integrate data sources into the FIS, an expert defines relationships between the mediator schema and the export schemas of the wrappers using a correspondence specification language (CSL, for example [CGH+ 94],[SS 96],[LNE 89]). Based on a common data model, correspondences explicitly specify and resolve conflicts caused by heterogeneous names, semantics and structures (see detailed classifications of heterogeneity in [SPD 92], [GSC 95]). From the information viewpoint, schemas and correspondences are the most important components in FIS.

The MoCA language[1] ([Bus 99]) introduced in chapter 2 is a CSL for conceptual design of FIS that is based on the ODMG data model ([CB 97]). In this paper we use ODMG schemas and MoCAs to analyze the connections of schemas and correspondences through evolution.

1.2 Schema Evolution of Federated Information Systems

As FIS are long-living information infrastructures, the system components must be flexible enough to cope with continuous change of components and configuration of the system. Whereas new sources can be easily integrated into the FIS by specifying new model correspondences, schema evolution requires modifications of correspondence specifications. We distinguish two kinds of evolution scenarios (see Figure 1):

- Global schema evolution: The change of the mediator schema causes the modification of all related correspondences to export schemas. The export schemas are not changed.
- Local schema evolution: The change of one export schema causes the modification of all correspondences related to the information source. MoCAs of other information sources are not affected.

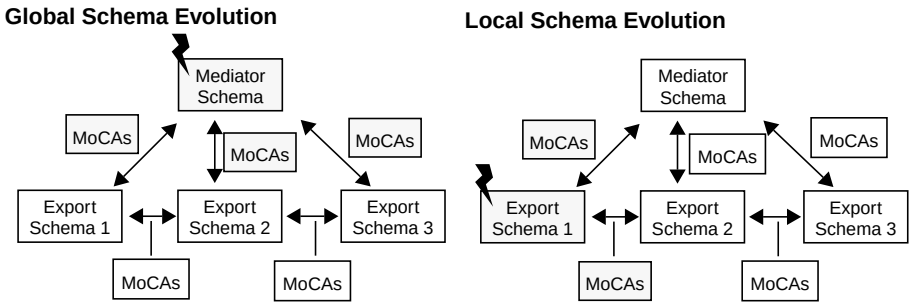


Fig. 1: Schema Evolution in FIS

---

[1] MoCA = Model Correspondence Assertion

Both evolution scenarios consists of the initial change of one schema and the propagation to related correspondence assertions. To guarantee consistency of the federation through evolution, we have to

- identify primitive modifications of schema and correspondence assertions,
- identify evolution transactions (e.g. meaningful groups of primitive modifications that occur together frequently), and
- give a formal definition of evolution operations, including applicability conditions and change propagation.

Our focus are connections between structural models – schemas – and model correspondences regarding their consistent evolution. Existing approaches (see [BKK+ 87], [Kol 99], [Ber 97], [KS 96], [BFG+ 98]) address the specification of effects of primitive evolution operations. [Kol 99] also specifies the effects on federated schemas in federated database systems. Going beyond these works, we define complex groups of primitive modifications. Those are important in FIS because structural modifications cannot be modeled as a combination of deletion and creation of model elements. The related correspondence assertions would be lost when one model element is deleted and have to be added manually later on.

1.3 Metamodel-based Specification of Evolution

In this paper we will use a metamodeling technique as our primary mechanism to encounter evolution. Both the ODMG data definition language and the MoCA language are modeled within the same dynamic logic framework. A remarkable feature of this approach is the ability to define the relation of intentional model entities and correspondence specification entities (see Figure 2). It allows not only for the description of structural relations among the modeling entities, but also for a formal definition of structural constraints and dynamic behavior of model evolution operations.

Our approach can be combined with similar metamodeling approaches in related contexts:

- [PBF 99] and [PK 99] use the same approach to specify constraints between the extensional and intentional model level and their consistent evolution, including both structural and behavioral aspects. This can be used in future work to specify evolution operations including rules to adapt existing informations sources and correspondence assertions.
- Many works use metamodels to define the semantics of modeling languages, for example of the UML [UML 99] ([EFL+ 98], [EFL+ 99]). The metamodeling approach and the separation of model level and correspondence level makes it easier to change the underlying common data model of the FIS and to compare or adapt the evolution specifications with other languages.

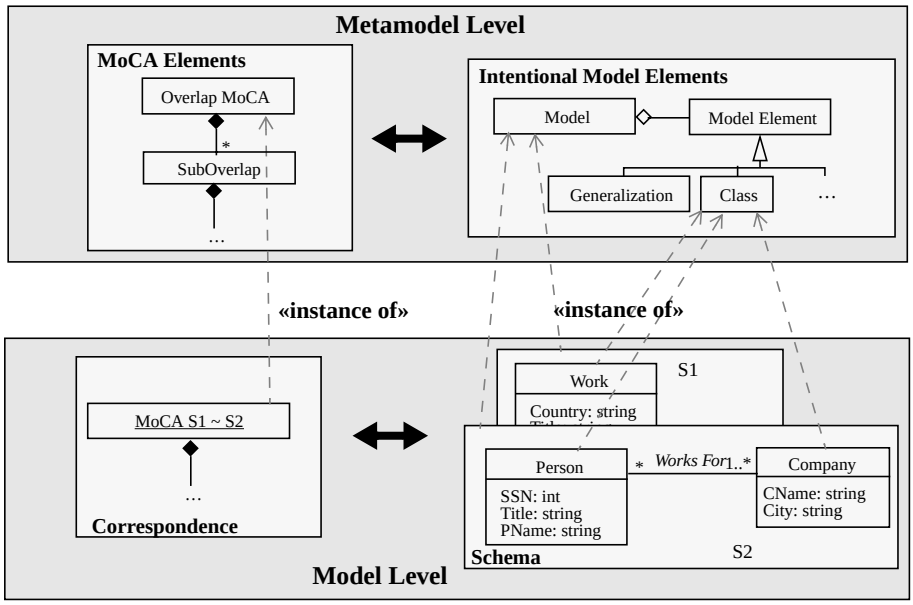


Fig. 2: Metamodeling Approach

Structure of the Paper

Chapter 2 gives a brief introduction to the MoCA language and the underlying metamodel. Chapter 3 classifies evolution operations in FIS and identifies useful evolution transactions. We describe the formalization of evolution operations with the focus on constraints between model intention and model correspondences. Chapter 4 concludes and gives an outlook to future work.

2 Model Correspondence Assertions

2.1 Purpose of MoCAs

MoCAs specify an overlap between two information sources or between mediator schema and export schema of one information source. Autonomy, heterogeneity, and overlapping universes of discourse are the main characteristics of information sources in FIS. Correspondence specifications are used for the three main tasks of a mediator that result from these characteristics:

- Schemas model the same real world objects in different manners. Therefore, MoCAs may specify *mapping rules between intensional descriptions*, for example to map different data types or representations.
- The *extensional overlap* between different data sources specified in a MoCA is relevant for identifying relevant sources in query processing and for query optimization.
- Data sources are typically overlapping in their extensions, i.e., they contain the description of same real world objects. The mediator has to identify same real world objects obtained from different sources in order to prevent redundant query results (object fusion, [PAG 96]). For this purpose, MoCAs may specify *properties for identifying semantically equivalent objects*.

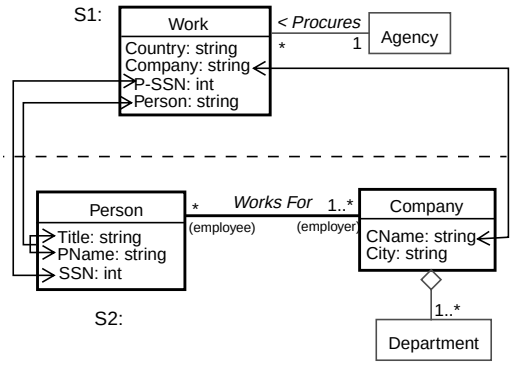
A MoCA relates one schema to one or more other schemas. A MoCA contains submodel correspondence assertions. A submodel correspondence assertion relates parts of the schemas, for example single object types. Because real world objects can be modeled in different ways, not only single object types, but also associated object types can constitute a submodel. Precisely, a submodel is defined as a view on the schema (the base schema), including projection on interesting types and selection of interesting extensions. View definitions use the object query language (OQL) of ODMG. Instances of the view are called *concepts*.

Each submodel correspondence assertion addresses the objectives described above. In the intensional part, it explicitly specifies mappings between intensional schema elements. If more than one attribute correspond to one attribute in the other schema, they are grouped by parentheses (Title and PName in the following example).

The extensional part contains the specification of properties that can be used for identifying corresponding objects and the specification of the extensional overlap between the submodels.

The MoCA language addresses logical heterogeneity, particularly descriptive conflicts (intensional part), extension conflicts (extensional part), and structural conflicts (grouping of schema elements). A deeper discussion of the language can be found in [Bus 99].

Example. Fig. 3 shows two data sources and the relating MoCA specification. Obviously, objects of type Work of S1 correspond to the associated objects of Person and Company in S2. The `elements` part of the MoCA defines these concepts. Corresponding objects are identified by the company name and the SSN. S2 contains data about all german companies and their employees. Therefore, we select only Work objects of germany to ensure identical extensions. Arrows indicate identified correspondences between attributes. The `intension` part of the MoCA specifies mappings between the attribute domains and defines the implicit attribute value of Country.




```
component overlap MoCA-Example
S1 ~ S2
   correspondence Working
   elements
     submodel Sub1:
       select Work w
       where w.Country = 'Germany'
     ~
     submodel Sub2:
       select Person p, p.employer c
   extension
     UoD relation Sub1 ≡ Sub2
     correspondence identifier
       w.Company ~ c.CName and
       w.P-SSN ~ p.SSN
   intension mappings
     w.Country = 'Germany'
     w.Company = c.CName
     w.P-SSN = p.SSN
     w.Person ≈ (p.Title, p.PName)
       with
       w.Person =
           p.Title || ' ' || p.PName
       and
       p.Title = firstWord(w.Person)
       and
       p.PName = secondWord(w.Person)
   end
end component overlap
```

Fig. 3: MoCA Example

2.2 The MoCA Metamodel

Figure 4 shows the metamodel of schemas and MoCAs. The metamodel consists of three levels:

- The *model level* shown at the bottom of the figure defines the model elements of the ODMG data model ([CB 97]). We use the definitions of the UML metamodel ([UML 99]) for the ODMG modeling concepts.
- The *MoCA concept level* in the middle of the figure defines the model elements and their properties that are used in a MoCA.
- The *MoCA correspondence level* specifies the correspondences between these concepts on intensional and extensional level.

The MoCA concept level connects the metamodel of ODMG schemas and the metamodel of MoCAs. The metamodel class Concepts represents a submodel definition. Instances are derived from the OQL statement. Because of structural heterogeneity, several model elements may constitute a submodel, such as Person and Company in our example before. These model elements are represented by ConceptElements. The name is the name given in the submodel definition.

ConceptProperty represents attributes and associations used in a MoCA. The property name does have a corresponding property of the base type on the model level. To capture 1:n attribute conflicts, a MoCA might also specify sets of corresponding properties.

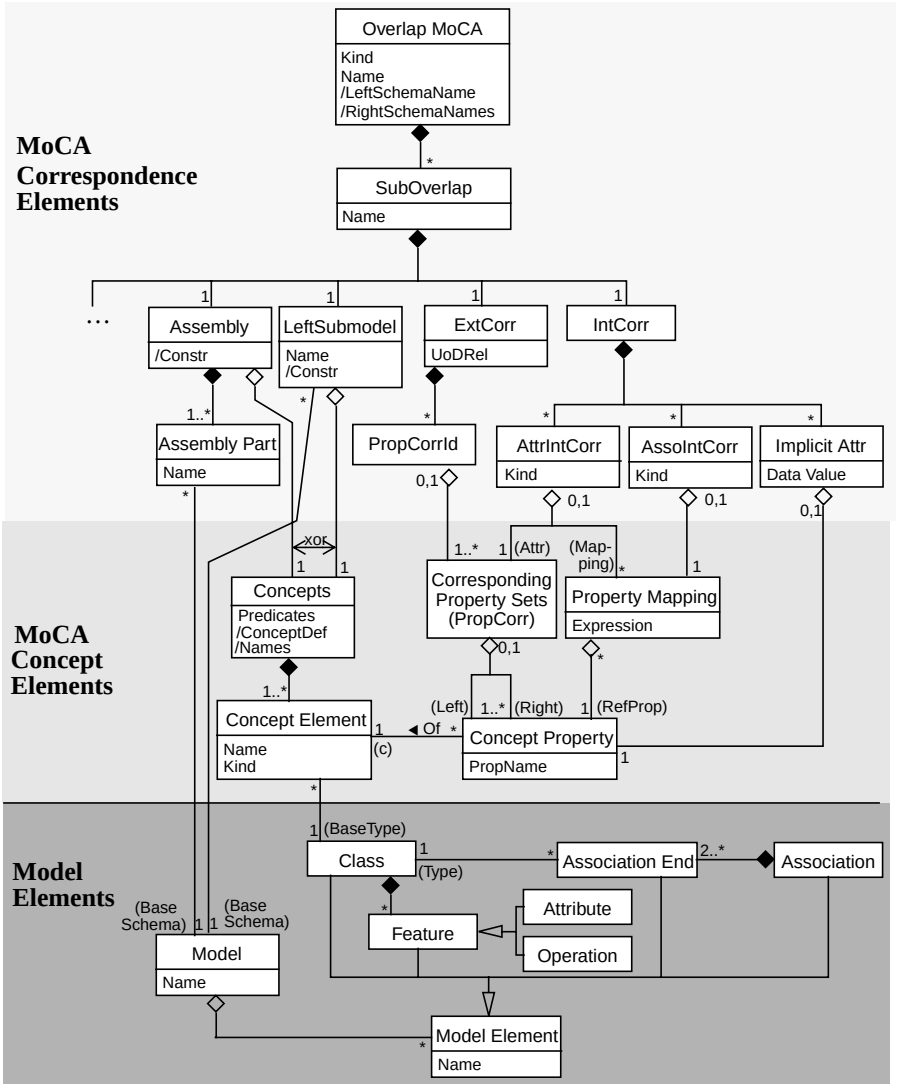


Fig. 4: Metamodel for Models and Model Correspondences

The MoCA correspondence level at the top of the figure models the MoCA with its submodel correspondence assertions. A submodel correspondence relates a LeftSubmodel with a submodel of one or more other schemas – an Assembly. ExtCorr represents the extensional part of a MoCA with the specified overlap of the universes of discourse and the identifying properties. IntCorr represents the intensional part with specified correspondences between attributes or outgoing associations and implicit attribute values such as Country. All correspondence specifications are characterized by a kind (of different types). It specifies for example if a mapping is bijective or if properties identify an extensional correspondence.

3 Metamodel-based Specification of Schema Evolution

We define evolution operations that enable the automatic adaptation of schema and correspondence level to guarantee consistency of the federation.

3.1 Classification of Evolution Operations

Evolution actions are classified in two categories (see Figure 5): Primitive Actions and Composite Actions. Primitive actions represent atomic modifications whereas composite actions represent groups of modifications that are applied together. A composite action represents an uninterruptable transaction. Consistency of the system must be guaranteed after the execution of the group of actions, but it may not be guaranteed during the execution.

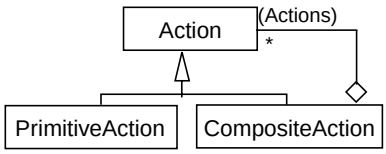


Fig. 5: Classification of Actions

Primitive Evolution Actions

We distinguish model evolution and MoCA evolution. From another point of view, evolution actions can be classified by the kind of modification. Based on the classification of [PK 99] we distinguish three groups (see Figure 6):

- Creation Actions insert a new element into either the model or the MoCA.
- Deletion Actions delete an existing element from either the model or the MoCA.

- Modification Actions modify an existing element from either the model or the MoCA.

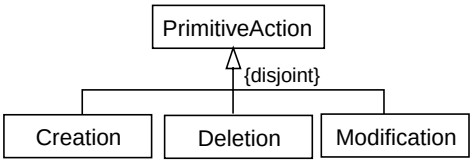


Fig. 6: Classification of Primitive Evolution Actions

A detailed classification of creation, deletion and modification actions in model evolution can be found in [BKK+ 87] or [Kol 99]. We extend this taxonomy and distinguish the following actions:

- Creation Actions
  Adding
  - an attribute
  - an operation
  - a class
  - an association
  - an association end
- Deletion Actions
  Deleting
  - an attribute
  - an operation
  - a class
  - an association
  - an association end
- Modification Actions
  - on class level
    Changing the
    - name of an attribute, operation, association, role, or class
    - domain of an attribute
    - signature of an operation: adding or deleting a parameter
    - implementation of an operation
    - cardinality of an association
  - on structural level
    - a class become supertype (subtype) of another class
    - a generalization/specialization relationship is deleted

In a similar way, we can define primitive evolution actions to add, delete or modify objects of the MoCA metamodel. They are used by experts to correct or to extend correspondence specifications and internally by the system to adapt the specifications after model evolution actions. We will give some examples in the following section.

Composite Evolution Actions

We only consider composite evolution actions initiated on model level, because we are interested in their effects on model correspondences. MoCAs cannot be derived automatically from schemas. Therefore, the composition of evolution actions to one uninterruptable transaction is useful if this leads in keeping model correspondences alive which would be lost otherwise. This situation occurs when modifying a schema because generally deletion and creation actions are combined and deletion actions cause deletion actions on MoCAs.

Based on our modeling experiences (see for example [KSW+ 95], [BK+ 00]) we propose six composite evolution actions for modifying the abstraction level of a schema. Examples are given in Figure 7.

1. SplitRecord: The record elements of one structured attribute become attributes of the class.
2. AttributeFusion: Several attributes of one class become elements of one structured attribute of the class. This is the reverse operation of SplitRecord.
3. SeparateAttributes: Several attributes of one class are moved to a new associated class.
4. IncludeAttributes: The attributes of an associated class are included in a class. This is the reverse operation of SeparateAttributes.
5. StructuralAbstraction: Two classes with one relating association build one new class. For example, S2 in our example in Figure 3 could be abstracted to one class 'Work'.
6. MoveAttributes: Attributes are related to another class.

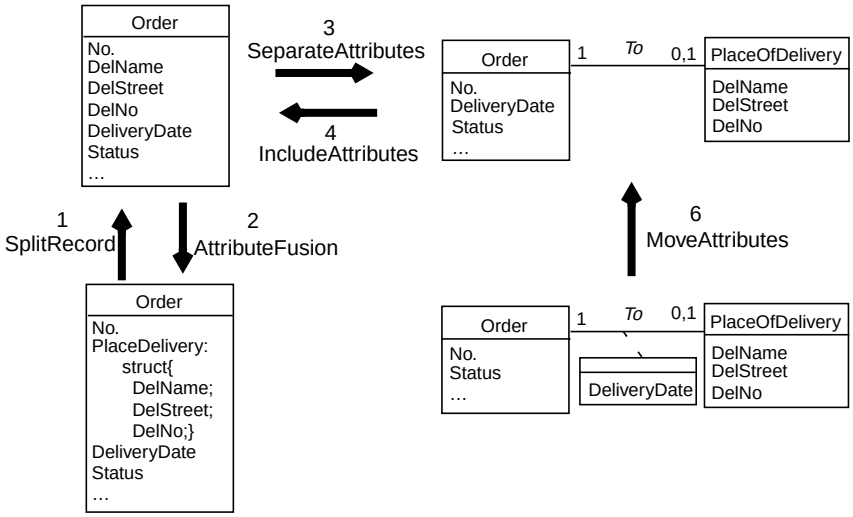


Fig. 7: Composite Evolution Actions – Examples

3.2 Formalization of Evolution Actions

The specification of evolution actions is based on the formalization of the metamodels. The dynamic logic signature[2] Σ_{ODMG} with a formula Φ_{ODMG} over Σ_{ODMG} represents the model level, Σ_{MoCA} and Φ_{MoCA} the MoCA level. The sorts in the signature correspond to model/MoCA elements (such as classes/concepts) and the action symbols represent modifications of the schema or the MoCA, respectively (evolution).

The model and the MoCA level are integrated by the FIS-theory, a first-order dynamic logic theory. It consists of a signature Σ_{FIS} and a set of axioms $\Phi_{FIS} = \Phi_{ODMG} \wedge \Phi_{MoCA} \wedge \Phi_{JOINT}$. The signature includes both the signature Σ_{ODMG} and the signature Σ_{MoCA}. It also contains a predicate symbol, 'Exists:Object' that defines the set of existing instances in each state.

The FIS-theory is interpreted as a set of transition systems (see [WB 98]). A transition system is a set of states with a set of transition relations on states. The domain for states is an algebra whose elements are model and MoCA elements. The set of transition relations includes transitions representing evolution of the models and correspondence assertions.
Each evolution action is defined by two formulas:

- **Necessary preconditions** to describe the applicability conditions of operations. The formula ($\langle op \rangle true \rightarrow cond$) states that the operation *op* is applicable only if the condition *cond* is true. Generally, preconditions result from static integrity constraints.
- **Sufficient postconditions** to describe the effect (direct effect and change propagation) of the operations. The formula ($[op]cond$) states that after the application of the operation *op* the condition *cond* is true.

These formulas may contain either model elements, or MoCA element or both. This allows us to define the effects from one level to the other. The specification of each evolution action has the following structure:

Action act
Precondition τ
Effect γ
Propagation δ

This represents the following dynamic formula: $(\langle act \rangle true \rightarrow \tau) \wedge ([act] (\gamma \wedge \delta))$.

---

2 A signature $\Sigma = ((\mathbf{S}, \leq), \mathbf{F}, \mathbf{P}, \mathbf{A})$ consists of a set of sort symbols **S**, a partial order relation between sorts $\leq$, a set of function symbols **F**, a set of predicate symbols **P**, and a set of Action symbols **A**.

We describe model evolution, MoCA evolution, and composite evolution with one example each. Thereby, we focus on the specification of dependencies between schema and correspondences.

Primitive Model Evolution Actions

Creation actions have no effects on model correspondence assertions, so that a propagation is only necessary in the model level.

The modification of model elements causes changes in the MoCAs that are based on these model elements. For example, change of property names have to be propagated to related concept properties. Other modifications like change of attribute domains mostly result in deleting the related correspondence specifications because the defined mappings do not hold anymore.

Deletion actions have similar effects: they have to be propagated to dependent correspondence elements.

Action setAttributeName(attr: Attribute, new: Name)

old = name(attr)

Precondition

Attribute names of the class are unique.

$\forall$a2: Attribute $\bullet$ (class(a2) = class(attr) $\rightarrow$ name(a2) $\neq$ new)

The name of an attribute cannot be the same as the name of an opposite association end.

$\forall$r1 $\in$ associationEnd(class(attr)), r2 $\in$ associationEnd(association(r1)) $\bullet$
(r1 $\neq$ r2 $\rightarrow$ name(r2) $\neq$ new)

Postcondition

name(attr) = new

Propagation

All concept properties in MoCAs that are based on attr are renamed.

$\forall$ce $\in$ conceptElement(class(attr)), cp $\in$ conceptProperty(ce) $\bullet$
propName(cp) = old $\rightarrow$ $\langle$setPropertyName(cp, new)$\rangle$ true

$\forall$ce $\in$ conceptElement(c) $\bullet$ $\langle$delConceptElement(ce)$\rangle$ true

MoCA Evolution Actions

Evolution actions on the MoCA level modify existing model correspondences. To emphasize the relationship to the model level, we show the specification of actions to add concept elements. In contrast to evolution on the model level, the dependency influences the preconditions of MoCA evolution actions. For clarity, we skip the modification of derived attributes like Names and ConceptDef.

Action addConceptElement(c: Concepts, n: Name, k: TCElemKind,
type: Name, m: Model, out el: ConceptElement)
TCElemKind = enum('independent', 'dependent', 'grouped')

Precondition

MoCA level: The concept definition exists, the concept element does not exist.
$Exists(c) \wedge \neg Exists(el)$

Model level:
The base schema exists.
$Exists(m)$

The base schema contains one class with the given type name.
$\exists!cl: Class \bullet model(cl) = m \wedge name(cl) = type$

Postcondition

The concept element exists in the context of the context definition.
$Exists(el) \wedge concepts(el) = c \wedge el \in conceptElement(c)$

The concept element is initialized. It contains no concept properties.
$name(el) = n \wedge kind(el) = k \wedge conceptProperty(el) = \emptyset$

The concept element is associated with the base type of the model level.
$\forall cl: Class \bullet (model(cl) = m \wedge name(cl) = type)$
$\rightarrow (baseType(el) = cl \wedge el \in conceptElement(cl))$

Composite Evolution Actions

Composite evolution actions consist of a set of (primitive) evolution actions which build one transaction. Usually the specification of the composite evolution action does not contain any postcondition, but only propagations to the primitive evolution actions of the transaction.

To explain the effects on model correspondences we consider the example of separating the attributes of delivery from the class Order given in Figure 7. This evolution action causes the following steps (see Figure 8):

- Adding the new class PlaceOfDelivery, the new association To and the attributes DelName, DelStreet and DelNo in the created class. As all attributes can be null, the cardinality of the new class is 0..1.
- Adapting the model correspondences that are based on separated attributes.
  - Adding a new concept element to the submodel definition. The concept element is based on the new class. The modification for our example is shown in the new MoCA at the bottom of Figure 8.
  - Moving the concept properties of separated attributes to the added concept element.
  - Modifying the extensional overlap if necessary.
    Elements of the changed submodel are associated objects of Order and PlaceOfDelivery. Single instances of Order which has no associated PlaceOfDelivery (all attributes are null) are not element of the submodel anymore. As a consequence, the extension of the submodel is smaller than before and the specified

overlap has to be modified. To keep the correspondence specification of single Order objects without PlaceOfDelivery, we copy the submodel correspondence before we add the new concept element to the submodel definition. As a result, we get two submodel correspondence assertions.

- Deleting the separated attributes in the old class.
  This action results in the deletion of correspondence specifications that are based on the 'old' attributes.

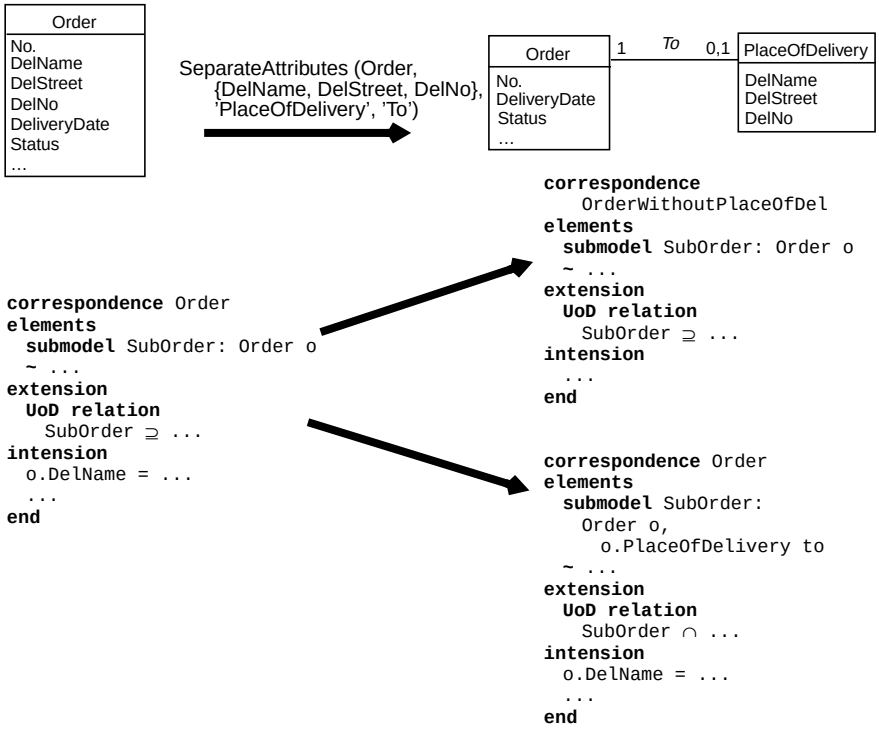


Fig. 8: SeparateAttributes – Example

The composite evolution action separateAttributes has the following specification. For clarity and briefness, we introduce some additional operations and predicates without detailed definition.

Action SeparateAttributes

(c: Class, attrs: set(Attribute), cName: Name, assName: Name)

newC: Class, newSub: SubOverlap

Precondition

All attributes are features of the class.

$\forall attr \in attrs \bullet attr \in feature(c)$

The name of the new class and the new association are unique.

$(\forall cl: Class \bullet model(cl) = model(c) \rightarrow name(cl) \neq cName) \ \wedge$

$(\forall ass: Association \bullet model(ass) = model(c) \rightarrow name(ass) \neq assName)$

…

Postcondition

Propagation

Model level: Adding the new class, association and attributes.

$(\langle addClass(model(c), cName, newC)\rangle \ true) \wedge$

$((\exists attr \in attrs \bullet \neg canBeNull(attr)) \rightarrow$

$\langle addAssociation(assName, \{(c, name(c), 1), (c2, cName, 1)\})\rangle \ true)$

$\wedge$

$((\forall attr \in attrs \bullet canBeNull(attr)) \rightarrow$

$\langle addAssociation(assName, \{(c, name(c), 1), (c2, cName, 0..1)\})\rangle \ true)$

$\wedge$

$(\forall attr \in attrs \bullet \langle addAttribute(c2, name(attr), type(attr))\rangle \ true)$

MoCA level: if separated attributes are used then

$\forall el \in conceptElement(c) \bullet attrs \cap conceptProperty(el) \neq \emptyset \ \rightarrow$

copy the submodel correspondence and change the extensional overlap if the cardinality is 0..1 and

$((\forall attr \in attrs, ec = extCorr(subOverlap(concepts(el))) \bullet$

$canBeNull(attr)) \ \rightarrow$

$(\langle copySubOverlap(subOverlap(concepts(el)),$

$name(subOverlap(concepts(el))) \ || \ 'Without' \ || \ cName,$

$newSub)\rangle \ true) \wedge$

$((UoDRel(ec) = 'Equivalence' \wedge isLeft(concepts(el))) \rightarrow$

$\langle setUoDRel(subOverlap(concepts(el)), 'Inclusion')\rangle \ true) \wedge$

…

$) \wedge$

add a new concept element and concept predicate and

$(\langle addConceptElement(concepts(el), assName, 'independent', newC,$

$model(newC))\rangle \ true) \wedge$

$(\langle addPredicate(concepts(el),$

$assName \ || \ '\in \ ' \ || \ name(el) \ || \ '.' \ || \ cName)\rangle \ true) \wedge$

move the concept properties of separated attributes

$(\forall cp \in conceptProperty \bullet (propName(cp) \in map(attrs, name) \rightarrow$

$\langle moveConceptProperty(c, newC, cp)\rangle \ true))$

Model level: Deleting the attributes in the initial class.

$(\forall attr \in attrs \bullet \langle delAttribute(attr)\rangle \ true)$

4 Conclusions

In this paper we defined a schema evolution mechanism for federated information systems (FIS) with formal semantics based on metamodeling and dynamic logic. From the information viewpoint, schemas and model correspondences – specifying semantic relationships between schemas – are the most important components in FIS. Schema evolution is an important aspect in long-living federations. As model correspondence assertions cannot be created automatically, mechanisms are necessary to adapt them consistently to evolving schemas.

Our metamodel-based approach provides one formal framework to specify the relationships between schemas and model correspondences. It allows not only for the formal definition of structural constraints, but also for the specification of the dynamic semantics of the modeled elements, that means the behavior of model elements and correspondences through evolution of the FIS. For each evolution action, we define semantically preconditions and describe the impact on other model elements (change propagation). In this way, consistency of the system through evolution is guaranteed.

In the second part we classified useful evolution actions. From a first classification of evolution scenarios in FIS, we defined primitive evolution actions and useful composite evolution actions for restructuring of a schema. They are transactions of several primitive operations and allow to keep existing correspondence assertions.
Future work will go in three directions:

- the detailed evaluation of such evolution transactions and their formal specification to complete our taxonomy;
- combining our approach with metamodel-based approaches for schema evolution on intentional and extensional level;
- embedding the formal framework within tools for specifying correspondence specifications and within mediator-based information systems (see [Oer 00]).

References

[Ber 97] P. Bergstein, *Maintenance of object-oriented systems during structural evolution*, Theory and Practice of Object Systems, Vol. 3, No. 3, John Wiley & Sons, 1997.

[Bus 99] S. Busse, *A Specification Language for Model Correspondence Assertions — Part I: Overlap Correspondences*, Technical Report, Forschungsberichte des Fachbereichs Informatik Nr. 99-8, TU Berlin, April 1999.

[BFG+ 98] E. Bertino, E. Ferrari, G. Guerrini, I. Merlo, *Extending the ODMG Object Model with time*, Proc. ECOOP'98, Lecture Notes in Computer Science 1445, 1998.

[BKK+ 87] J. Banerjee, W. Kim, H.J. Kim, H.F. Korth, *Semantics and Implementation of Schema Evolution in OODB*, Proc. 5th ACM SIGMOD Conference on Management of Data, ACM, pp. 311-322, 1987.

[BK+ 00] S. Busse, R.-D. Kutsche et al., *Modellierung informationslogistischer Anwendungen – Projektbericht*, Forschungsberichte des Fachbereichs Informatik Nr. 2000-9, TU Berlin, 2000.

[BKLW 99] S. Busse, R.-D. Kutsche, U. Leser, H. Weber, *Federated Information Systems – Concepts, Terminology and Architectures*, Technical Report, Forschungsberichte des Fachbereichs Informatik Nr. 99-9, TU Berlin, April 1999.

[CB 97] R.G.G. Cattell, D.K. Barry (eds.), *The Object Database Standard: ODMG 2.0*, Morgan Kaufmann, 1997.

[CGH+ 94] S. Chawathe, H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. D. Ullman, J. Widom, *The TSIMMIS Project: Integration of Heterogeneous Information Sources*, 16th Meeting of the Information Processing Society of Japan, pp. 7-18, Tokyo, Japan, 1994.

[EFL+ 98] A. Evans, R. France, K. Lano, B. Rumpe, *Developing the UML as a formal modeling notation*, in: Muller, Bezivin (eds.), *UML'98 Beyond the notation*, International Workshop, Lecture Notes in Computer Science LNCS 1618, Springer, 1998.

[EFL+ 99] A. Evans, R. France, K. Lano, B. Rumpe, *Towards a core metamodelling semantics of UML*, in: H. Kilov (ed.), *Behavior specifications of business and systems*, Kluwer Academic Publishers, 1999.

[GSC 95] M. Garcia-Solaco, F. Saltor, M. Castellanos, *Semantic Heterogeneity in Multidatabase Systems*, in: O.A. Bukhres, A.K. Elmagarmid (eds.), *Object-Oriented Multidatabase Systems*, Prentice Hall, 1996; pp. 129-202, 1995.

[Kim 95] W. Kim (ed.), *Modern Database Systems*, Addison-Wesley, 1995.

[Kol 99] S. Kolmschlag, *Schemaevolution in Föderierten Datenbanksystemen*, Dissertation D 466, Universität-GH Paderborn, C-LAB Publication, Band 2, Shaker Verlag, 1999.

[KCGS 95] W. Kim, I. Choi, S. Gala, M. Scheevel, *On Resolving Schematic Heterogeneity in Multidatabase Systems*, in: [Kim 95], pp. 521-550, 1995.

[KS 96] F. Kesim, M. Sergot, *A logic programming framework for modeling temporal objects*, IEEE Transactions on Knowledge and Data Engineering, Vol. 8, No. 5, Oct. 1996.

[KSW+ 95] R.-D. Kutsche, C. Schöning, S. Waßerroth et al., *Informationsmodellierung im Rahmen eines Umweltinformationssystems*, Forschungsberichte des Fachbereichs Informatik Nr. 95-17, TU Berlin, April 1995.

[LNE 89] J.A. Larson, S.B. Navathe, R. Elmasri, *A Theory of Attribute Equivalence in Databases with Applications to Schema Integration*, IEEE Transactions on Software Engineering, Vol. 15, No. 4, pp. 449-463, Apr. 1989.

[Oer 00] L. Oergel, *Viewpointübergreifende Konzeption eines evolutionsfähigen Informationsmediators*, Diplomarbeit, TU Berlin, FB Informatik, Juli 2000.

[PAG 96] Y. Papakonstantinou, S. Abiteboul, and H. Garcia-Molina, *Object Fusion in Mediator Systems*, in: T.M. Vijayaraman, A.P. Buchmann, C. Mohan, N.L. Sarda (eds.), *22nd Conf. on Very Large Databases*, VLDB, Mumbai (Bombay), India, 1996.

[PBF 99] C. Pons, G. Baum, M. Felder, *Foundations of Object-oriented Modeling Notations in a Dynamic Logic Framework*, in: T. Polle, T. Ripke, K. Schewe (eds.), *Fundamentals of Information Systems*, Kluwer Academic Publishers, chapter 1, 1999.

[PK 99] C. Pons, R.-D. Kutsche, *Model evolution and system evolution*, Proc. CACIC'99, Universidad Nacional del Centro de la Provincia de Buenos Aires, Argentinia, Nov. 1999.

[SL 90] A.P. Sheth, J.A. Larson, *Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases*, ACM Computing Surveys, Vol. 22, No. 3, pp. 183-236, Sep. 1990.

[SPD 92] S. Spaccapietra, C. Parent, Y. Dupont, *Model Independent Assertions for Integration of Heterogeneous Schemas*, The VLDB Jounal, Vol. 1, No. 1, pp. 81-126, Jul. 1992.

[SS 96] I. Schmitt, G. Saake, *Schema Integration and View Generation by Resolving Intensional and Extensional Overlappings*, in: K. Yetongnon, S. Hariri (eds.), Proc. 9th ICSA Int. Conf. on Parallel and Distributed Computing Systems (PDCS'96), pp. 751-758, Sep. 1996.

[UML 99] Object Management Group, *The Unified Modeling Language (UML) Specification – Version 1.3*, available at http://www.omg.org/, 1999.

[Wie 92] G. Wiederhold, *Mediators in the Architecture of Future Information Systems*, IEEE Computers, Vol. 25, No. 3, pp. 38-49, Mar. 1992.

[WB 98] R. Wieringa, J. Broersen, *Minimal Transition System Semantics for Lightweight Class and Behavior Diagrams*, in: M. Broy, D. Coleman, T. Maibaum, B. Rumpe, *PSMT Workshop on Precise Semantics for Software Modeling Techniques*, TUM-I9803, TU München, 1998.